



TITLE:

Program Schemas without Gotos (アルゴリズムにおける証明論)

AUTHOR(S):

KANAYAMA, YUTAKA

CITATION:

KANAYAMA, YUTAKA. Program Schemas without Gotos (アルゴリズムにおける証明論). 数理解析研究所講究録 1975, 236: 31-37

ISSUE DATE:

1975-05

URL:

<http://hdl.handle.net/2433/105508>

RIGHT:

PROGRAM SCHEMAS WITHOUT GOTOS

Yutaka Kanayama

Department of Computer Science

The University of Electro-Communications

Chofugaoka, Chofu, Tokyo, JAPAN

Phone. 0424-83-2161

ABSTRACT

A programming system L for non-deterministic program schema is introduced. The principal features of L are as follows:

- (1) All programs in L have two exits as subroutines in SNOBOL do.
- (2) The branching function is realized by connectives \cdot and $+$, and a duality is observed between them.
- (3) The looping function is realized by recursive calls which is represented by a naming operator π .
- (4) The fourth connective $(-)$ has the exit-exchanging effect which has no equivalents in conventional programming languages.
- (5) All predicate type operations in L may have side effects.

In a sense, L is a proposal for goto-less programming. For example, two programs if p then α else β and while p do α are translated into L as follows: $p\alpha + \beta$ and $\pi x(p\alpha x + 1)$. A program $\pi x(pa + q + bx)$, however, has no equivalents in D-chart.

The meaning of a program \bigwedge_{α} is defined from its computation $|\alpha|$, which is a pair of simple deterministic languages. Hence the equivalence problem in L is solvable.

0. INTRODUCTION

Ianov introduced an abstract model of computer programs and showed that the equivalence problem among them is solvable.[5] Ianov schemas permit, however, unlimited use of GOTOs which are considered undesirable recently. In this paper we present a GOTO-less programming language system L in which loops are expressed by recursive calls.

In Section 1, we present the syntax of L and a computation $|\alpha|$ of a program α . It is easy to see that $|\alpha|$ is a pair of simple deterministic languages.[7]

The semantics is given in Section 2. ^{Since} The meaning of α is completely determined by $|\alpha|$, the equivalence problem in L is solvable.

In appendices, the relations between our system and others are discussed.

1. PROGRAMS AND THEIR COMPUTATIONS

First, we introduce the syntax of L. We use three kinds of basic symbols and variables.

$A_0 = \{0\}$ is the singleton set of a null exit symbol.

$A_1 = \{1, a, b, c, \dots\}$ is the set of single exit symbols.

$A_2 = \{p, q, r, \dots\}$ is the set of double exit symbols.

$V = \{x, y, z, \dots\}$ is the set of variables.

A program in L is a string constructed by basic symbols, variables, \cdot , $+$, π (naming operator) and parentheses:

- (1) A basic symbol or a variable is a program.
- (2) If $x \in V$ and α and β are programs, then so are $(\alpha \cdot \beta)$, $(\alpha + \beta)$, $(-\alpha)$ and $(\pi x \alpha)$.
- (3) A string is a program only if it can be shown to be a program by (1) and (2).

In a program $(\pi x \alpha)$, the occurrence x is called a name and α a scope of the name. An occurrence of a variable x is said to be bound if it is a name or it is in a scope of the same name x ; otherwise, free. A program is said to be closed if it has no free occurrences of variables. The notion of "normal form program" supports the definition above that a free variable in a scope is bound by a name.[1][3]

At this point, we stipulate some conventions to avoid the use of parentheses and connectives in writing programs. First, we may omit the outer pair of parentheses in a program. Second, the connectives are ordered as follows: $-$, π , \cdot , $+$. Third, $(-\alpha)$ may be written as $\bar{\alpha}$. Fourth, dots may be omitted. Then $p\bar{x} + a$ stands for $((p \cdot (-x)) + a)$. *Fifth, $\alpha * \beta * \gamma$ denotes $((\alpha * \beta) * \gamma)$ for $*$ = \cdot or $+$.* Thereafter, $*_1, *_2, \dots$ and $*$ stand for \cdot or $+$.

In order to define the meaning of a program, we may construct an abstract machine with a push-down stack which executes non-deterministic computations under a specific interpretation. We adopt, however, another way because it is easier for us to utilize a well known result in formal language theory.

Two alphabets Σ and Σ_V denote the sets $\{a. \mid a \in A_1 - \{1\}\}$ $\{p., p_+ \mid p \in A_2\}$ and $\Sigma \cup \{x., x_+ \mid x \in V\}$ respectively. The set of all words generated by an alphabet Z is denoted by Z^* and the empty word, λ . If $W_1, W_2 \subseteq Z^*$, then $W_1 W_2 (\subseteq Z^*)$ denotes the set $\{w_1 w_2 \mid w_1 \in W_1, w_2 \in W_2\}$. Let $W., W_+ \subseteq \Sigma_V^*$, $W = (W., W_+)$ and $w \in \Sigma_V^*$. Then $w[W/x]$ means the set of all words obtained from w by replacing each occurrence of $x.$ in w by some $w. \in W.$ and each occurrence of x_+ in w by some $w_+ \in W_+$; i.e., $w[W/x] = \{v_0 w_{*1} v_1 w_{*2} \dots w_{*k} v_k \mid v_0 x_{*1} v_1 x_{*2} \dots x_{*k} v_k = w \wedge v_0, \dots, v_k \in (\Sigma_V - \{x., x_+\})^* \wedge *1, \dots, *k \in \{., +\} \wedge w_{*1} \in W_{*1} \wedge \dots \wedge w_{*k} \in W_{*k}\}$. Furthermore, if $W.', W_+' \subseteq \Sigma_V^*$ and $W' = (W.', W_+')$, then we stipulate that $W'[W/x] = (\bigcup_{w \in W.'} w[W/x], \bigcup_{w \in W_+'} w[W/x])$. If α is a program, then a computation of α , $|\alpha| = (|\alpha|., |\alpha|_+)$ is defined as follows ($|\alpha|.$ and $|\alpha|_+$ are called a dot computation and plus computation of α respectively.):

$$|0| = (\phi, \phi)$$

$$|1| = (\{\lambda\}, \phi)$$

$$|a| = (\{a.\}, \phi), \quad \text{if } a \in A_1 - \{1\}$$

$$|p| = (\{p.\}, \{p_+\}), \quad \text{if } p \in A_2$$

$$|x| = (\{x.\}, \{x_+\}), \quad \text{if } x \in V$$

$$|\alpha \cdot \beta| = (|\alpha|., |\beta|., |\alpha|_+ \cup (|\alpha|., |\beta|_+))$$

$$|\alpha + \beta| = (|\alpha|., |\alpha|_+ \cup (|\alpha|_+, |\beta|.), |\alpha|_+, |\beta|_+)$$

$$|\bar{\alpha}| = (|\alpha|_+, |\alpha|.)$$

$$|\pi x \alpha| = \bigcup_{n=0}^{\infty} |\alpha|_x^n, \quad \text{where} \quad \begin{cases} |\alpha|_x^0 = (\phi, \phi), \\ |\alpha|_x^{n+1} = |\alpha| [|\alpha|_x^n / x]. \end{cases}$$

In fact $|\alpha|., |\alpha|_+ \subseteq (\Sigma \cup \{x., x_+ \mid x \text{ is a free variable in } \alpha\})^*$.

Hence, if α is closed, then $|\alpha|., |\alpha|_+ \subseteq \Sigma^*$.

Example 1.1 $|pa + b|. = \{p.a., p_+b.\}$, $|pa + b|_+ = \phi$.
 $|pax + 1|. = \{p.a.x., p.a.x_+, p_+\}$, $|pax + 1|_+ = \phi$.
 $|\pi x(pax + 1)|. = \{p_+, p.a.p_+, p.a.p.a.p_+, \dots\}$, $|\pi x(pax + 1)|_+ = \phi$.
 $|px + q|. = \{p.x., p.x_+q., p_+q.\}$, $|px + q|_+ = \{p.x_+q_+, p_+q_+\}$.
 $|\pi x(px + q)|. = \{p_+q., p.p_+q., p.p_+q_+q., p.p.p_+q., p.p.p_+q_+q., \dots\}$,
 $|\pi x(px + q)|_+ = \{p_+q_+, p.p_+q_+q_+, p.p.p_+q_+q_+q_+, \dots\}$.

Korenjak and Hopcroft introduced the class of "simple deterministic languages" in their paper[7]. Now we adopt an extended definition that the singleton set $\{\lambda\}$ also is said to be simple deterministic.

Theorem 1.1 For any α , $|\alpha|.$ and $|\alpha|_+$ are simple deterministic.

Theorem 1.2 It is undecidable whether $|\alpha|_* \cap |\beta|_* = \phi$ for arbitrary α and β , for each $*$.

2. SEMANTICS

In this section, we describe how nondeterministic computations of a program go on a specific domain.

Let D be an arbitrary nonempty set and $\mathcal{F}(D)$ the class of all partial functions: $D \rightarrow D$. An interpretation~~in~~_I in L is a pair (D, θ) , where θ is a function: $\Sigma_V \rightarrow \mathcal{F}(D)$. It is extended to $\Sigma_V^* \rightarrow \mathcal{F}(D)$ as follows:

$$\begin{cases} \theta(\lambda) = \lambda u u \text{ (= the identity function on } D), \\ \theta(wc) = \lambda u [\theta(c)(\theta(w)(u))], \text{ if } w \in \Sigma_V^* \text{ and } c \in \Sigma_V, \end{cases}$$

where $\theta(w)(u) = \text{undefined}$ implies $\theta(c)(\theta(w)(u)) = \text{undefined}$.

If $W \subseteq \Sigma_V^*$ and $u \in D$, then $\theta(W)(u)$ denotes the set $\{\theta(w)(u) \mid w \in W, \theta(w)(u) = \text{defined}\}$. We write $\alpha =_I \beta$ if $\theta(|\alpha|_*)(u) = \theta(|\beta|_*)(u)$ for any u and $*$. Furthermore we write $\models \alpha = \beta$ if $\alpha =_I \beta$ for any I .

Theorem 2.1 $\models \alpha = \beta$ iff $|\alpha| = |\beta|$.

Theorem 2.2 It is decidable whether $\models \alpha = \beta$ for any α and β .

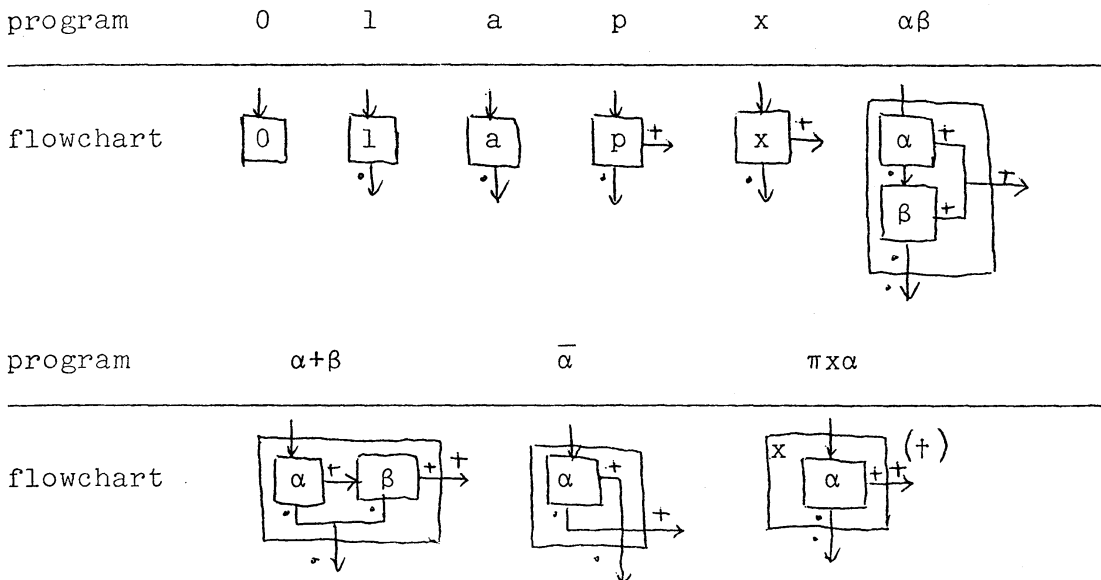
REFERENCES

- [1] Cooper, D.C., Programs for mechanical program verification, Machine Intelligence 6, Edinburgh U. Press, pp43-59.
- [2] Dahl, O.-J., E.W. Dijkstra and C. A. R. Hoare, Structured programming, Academic Press, New York, 1972.
- [3] Engeler, E., Structure and meaning of elementary programs, Symp. on semantics of algorithmic languages, pp 89-101, Springer, 1971.
- [4] Farber, D. J., R. E. Griswold and I. P. Polonsky, The SNOBOL 3 programming language, BSTJ, 1966, pp 895-929.
- [5] Ianov, I., The logical schemes of algorithms, in Problems of cybernetics, Pergamon Press, pp 82-140, 1960.
- [6] Knuth, D. E. and R. W. Floyd, Notes on avoiding "Go to" statements, Information Processing Letters 1, pp 23-31.
- [7] Korenjak, A. J. and J. E. Hopcroft, Simple deterministic languages, Record of SWAT Symp., 1966, pp 36-46.

APPENDICES

A. TRANSLATION OF PROGRAMS INTO FLOWCHARTS

For any program α , its flowchart equivalent has zero, one or two exits as subroutines in SNOBOL do.[4]



(+) Free variables x in α are regarded as equal to the whole program α .

B. TRANSLATION OF D CHARTS INTO PROGRAMS

Any D-chart[2] is translated in L as follows:

- (1) $a \rightarrow a,$
- (2) $\alpha \text{ then } \beta \rightarrow \alpha\beta,$
- (3) $\text{if } p \text{ then } \alpha \text{ else } \beta \rightarrow p\alpha + \beta,$
- (4) $\text{while } p \text{ do } \alpha \rightarrow \pi x(p\alpha x + 1).$

Note that if α is of this type, then α contains at most one variable x and $|\alpha|_+ = \phi$. It is impossible to convert any program in L into D-chart. For example, $\pi x(pa + q + bx)$ has no flowchart equivalents.[6]